Implementation of Johnson's APSP algorithm using different heaps

Rishabh Jain

AIM- Implementation of Johnson's APSP algorithm using an array based heap, Binary heap, Binomial heap and Fibonacci heap and analyze the time complexity of the algorithm corresponding to these.

Algorithm description-

In Johnson's APSP algorithm, we add a new vertex to the graph, connect it to all other vertices with edge weights 0 and run Bellman ford from it to find the shortest distance from the new vertex to all the vertices. Then, all the edge weights are re-weighted using the values. This is done so that all the edge weights become positive and Dijkstra algorithm could be used from all the vertices to find APSP. **Thus, complexity of the algorithm is O(V*(Dijkstra) + Bellman Ford) or O(V*(Dijkstra) + VE).**

Comments about the random graphs used in this experiment-

Adjacency matrix for large random graphs in this experiment were generated using pseudo random number generators. Thus-

1. **The graphs are dense graphs**. **E (number of edges) is somewhere around O($V^{1.3}$)** (approximately). Pseudo random functions were used to decide whether to create an edge or not. If yes, then another random function decided the edge weight.

2. Our **main motive in the assignment was to study the effect of the different heaps on the complexity of the algorithm**. Thus, **V times Dijkstra step of the algorithm is of highest importance**. **The test graphs are such that the complexity of Bellman ford step is not worse than O(V^2)**. Otherwise, Bellman Ford step will become the dominating term in the complexity and we will not be able to analyze the effect of using different heaps properly.

1. Array based implementation-

Algorithm for array based Dijkstra's algorithm-

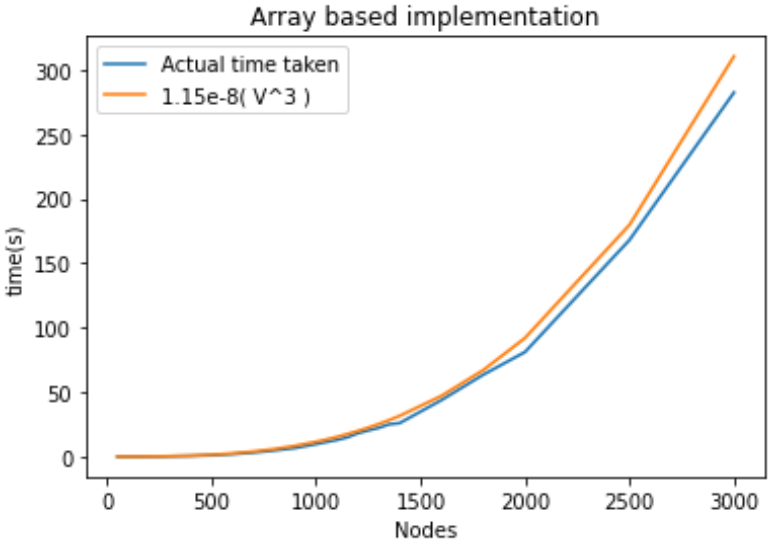
1. Keep an array of size V, which contains the distance of the vertex and the whether the vertex has been visited or not.
2. Mark source's distance=0.
3. In the dijsktra algorithm, we traverse the array to find an unvisited vertex at minimum distance in O(V) time. Then, we relax its edges until no vertex is available in O(E) time.

Theoretical time complexity of array based Johnson's algorithm-

1. For Dijkstra from one vertex, the time complexity is O(V^2+E)
2. Hence for Dijkstra algorithm executed from all the vertices, time complexity is O(V^3+VE)

Hence theoretical time complexity for array based Johnson algorithm is
O(V^3+VE) + O(VE) = O(V^3).

Experimental findings-



From the graph, we can observe that the time taken by the algorithm is **always upper bounded by 1.15e-8(V^3), which is O(V^3).**

Hence it is proved experimentally that the time complexity for array based Johnson algorithm is O(V^3).

2. Binary Heap based implementation-

Algorithm for Binary Heap based Dijkstra's algorithm-

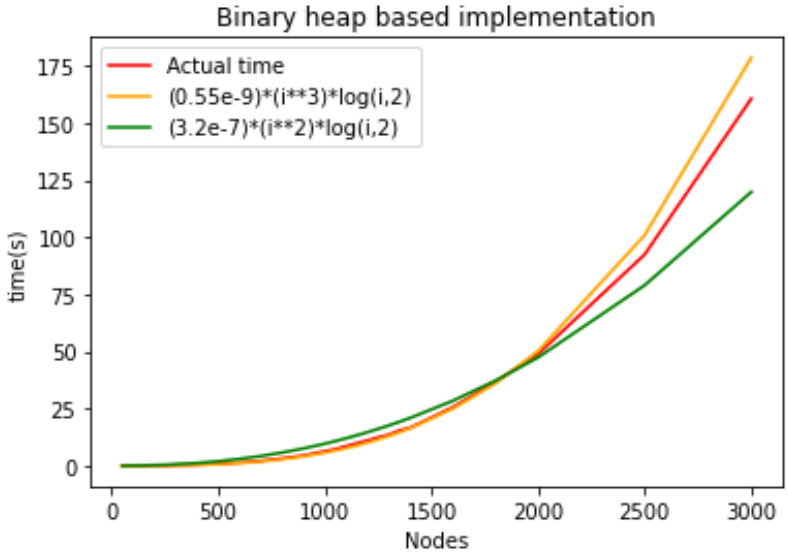
1. Perform make heap operation on an array of V nodes according to distance from source. This has been optimized to O(logV) from O(VlogV) in my code since we know that initially only the source has distance=0 and all other nodes are considered at infinity.
2. Then, relax operations are performed from the vertices.
3. Extract_min() in performed, which is O(logV). Next, for every relax operation decrease_key() is called, which is O(logV).
4. Hence, for one Dijkstra run, total complexity is O((V+E)logV).

Theoretical time complexity of Binary Heap based Johnson's algorithm-

1. For Dijkstra from one vertex, the time complexity is O((V+E)logV).
2. Hence for Dijkstra algorithm executed from all the vertices, time complexity is O((V^2+VE)logV).

Hence theoretical time complexity for Binary Heap based Johnson algorithm is O((V^2+VE)logV).

Experimental findings-



1. Here, we observe that the plot is **asymptotically above V^2LogV and below V^3LogV.**
2. We observe that the **theoretical complexity of a binary heap based implementation is $O((V^2+VE)logV)$.**
3. For a **dense graph, $E=O(V^2)$.** Thus for $E=O(V^2)$, complexity = **$O(V^3logV)$.**
4. For a **sparse graph**, complexity of the algorithm is **$O(V^2LogV)$.**
5. **Since E can be large, it can have significant effect on the complexity** and thus can increase the complexity of the algorithm.

Hence, here we obtain the complexity slightly worse than $O(V^2LogV)$ but better than $O(V^3LogV)$.

2. Binomial Heap based implementation-

Algorithm for Binomial Heap based Dijkstra's algorithm-

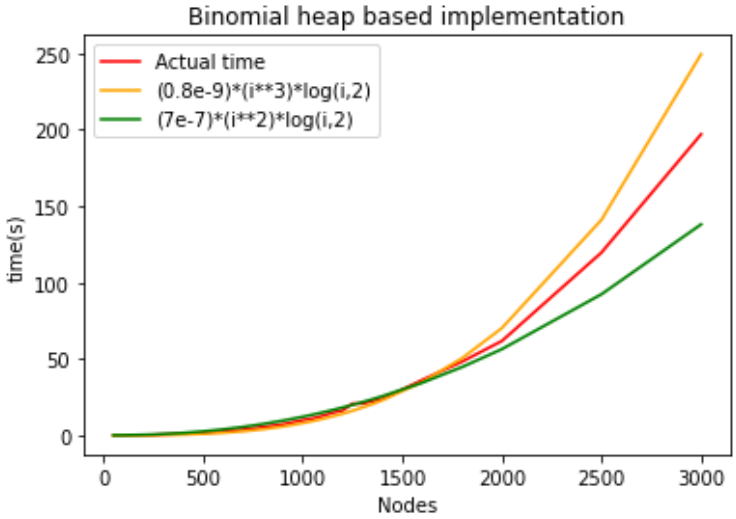
1. Perform make heap operation on an array of V nodes according to distance from source. This operation is O(V) since initially we have a root list of V elements which need to be merged to make a binomial heap
2. Then, relax operations are performed from the vertices, starting from the source.
3. Extract_min() in performed, which is O(logV). Next, for every relax operation decrease_key() is called, which is O(logV).
4. Hence, for one Dijkstra run, total complexity is O((V+E)logV).

Theoretical time complexity of binomial heap based Johnson's algorithm-

1. For Dijkstra from one vertex, the time complexity is O((V+E)logV).
2. Hence for Dijkstra algorithm executed from all the vertices, time complexity is O((V^2+VE)logV).

Hence theoretical time complexity for Binomial Heap based Johnson algorithm is O((V^2+VE)logV).

Experimental findings-



1. Here, we observe that the plot is **asymptotically above V^{2}LogV and below V^{3}LogV.**
2. We observe that the theoretical complexity of binary heap based implementation is O((V^2+VE)logV).
3. For a **dense graph, E=O(V^2).** Thus for E=O(V^2), **complexity = O(V^{3}logV**).
4. For a **sparse graph, complexity** of the algorithm **is O(V^{2}LogV).**
5. Since E can be large, it can have significant effect on the complexity and thus can increase the complexity of the algorithm.

Hence, here we obtain the complexity slightly worse than O(V^{2}LogV) but better than O(V^{3}LogV).

4. Fibonacci Heap based implementation-

Algorithm for Fibonacci Heap based Dijkstra's algorithm-

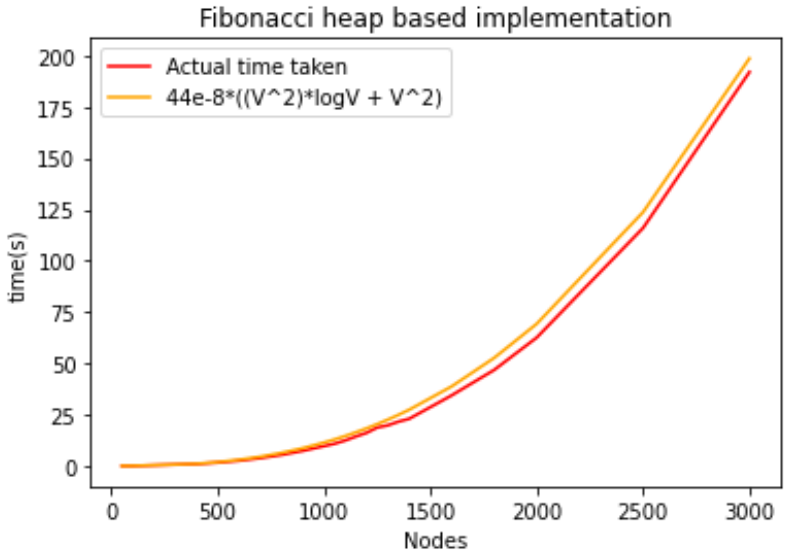
1. Perform make heap operation on an array of V nodes according to distance from source. This operation is O(V) and involves creation of a root list.
2. Then, relax operations are performed from the vertices, starting from the source.
3. Extract_min() in performed, which is O(logV) amortized. Next, for every relax operation decrease_key() is called, which is O(1) amortized.
4. Hence, for one Dijkstra run, total complexity is O(VlogV + E).

Theoretical time complexity of binomial heap based Johnson's algorithm-

1. For Dijkstra from one vertex, the time complexity is O(VlogV + E).
2. Hence for Dijkstra algorithm executed from all the vertices, time complexity is $O(V^2 logV+VE)$.

Hence theoretical time complexity for Fibonacci Heap based Johnson algorithm is $O(V^2 logV+VE)$.

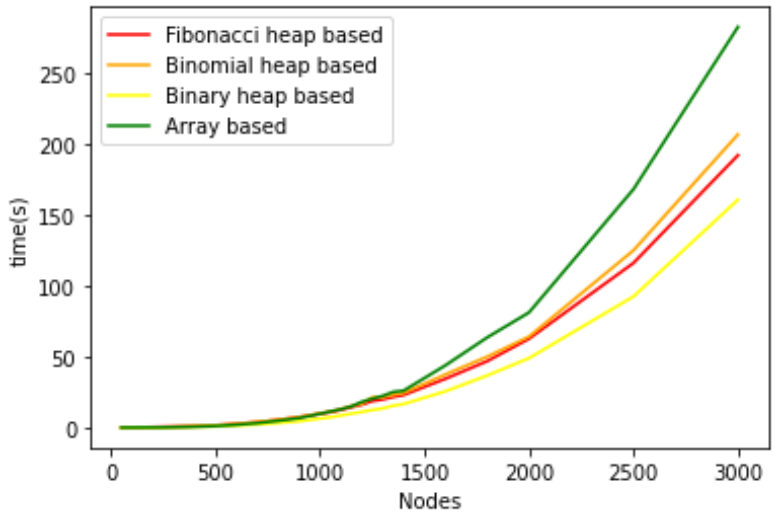
Experimental findings-



Here we observe that actual time taken by the algorithm **is upper bounded by $44e\text{-}8(V^2 logV+V^2)$,** which is $O(V^2 logV)$.

Hence the time complexity for Fibonacci Heap based Johnson algorithm is proven to be $O(V^2 logV)$.

Observations from this experiment-



(This graph is NOT intended to compare the complexities of the 4 implementations by plotting them on the same plot. It is plotted just to compare the raw execution time.)

The main takeaway points from the above graph is-

1. The **array based implementation takes the most execution time.** Theoretically it should also take the worst time among all the implementations of $O(V^3)$.
2. **Binomial heap is slower than Fibonacci heap as expected**. Both are faster than array based implementation.
3. One may observe that binary heap based implementation is fastest. However, its time complexity is worse than Binomial heap and Fibonacci heap based algorithms as shown previously. It is simply taking less time since its implementation is much simpler as compared to Binomial heaps and Fibonacci heaps. Thus raw execution time of the different implementations cannot be compared.

Conclusions from this experiment-

Fibonacci heap implementation is proven to have the best time complexity among all other implementations.

However, it is a computationally expensive heap on generic hardware/common programming languages and thus takes more raw execution time.

----------END----------